Acta et Commentationes, Sciences of Education, nr. 2(40), 2025 ISSN 1857-0623
p. 28-38 E-ISSN 2587-3636

CZU: 004.43’232C#:004.451.3 DOI: 10.36120/2587-3636.v40i2.28-38

ASPECTE TEORETICE ȘI APLICATIVE PRIVIND TRATAREA EXCEPȚIILOR ÎN LIMBAJUL DE PROGRAMARE C#

Ala GASNAŞ, dr., conf. univ.
https://orcid.org/0000-0002-7174-7027
Universitatea Pedagogică de Stat „Ion Creangă” din Chișinău

Rezumat. Acest articol oferă o analiză detaliată a mecanismului `try-catch-finally` în limbajul de programare C#, evidențiind funcționalitatea fiecărei componente și ilustrând aplicabilitatea acestora prin exemple relevante pentru diverse situații de eroare. Sunt examinate diferențele conceptuale și funcționale dintre excepțiile generale și cele specifice, precum și contextul adecvat de utilizare a instrucțiunii `throw` pentru generarea excepțiilor personalizate. În plus, sunt abordate principiile și practicile recomandate în gestionarea excepțiilor, cu scopul de a susține dezvoltarea unor aplicații stabile, predictibile și conforme cu standardele de calitate ale programării moderne.
Cuvinte cheie: mecanismul de tratare a excepțiilor, limbajul de programare C#, gestionarea erorilor, excepții, tipuri de excepții.

THEORETICAL AND PRACTICAL ASPECTS OF EXCEPTION HANDLING IN THE C# PROGRAMMING LANGUAGE

Abstract. This article provides a detailed analysis of the `try-catch-finally` mechanism in the C# programming language, highlighting the functionality of each component and illustrating their applicability through relevant examples for various error-handling scenarios. It examines the conceptual and functional differences between general and specific exceptions, as well as the appropriate use of the `throw` statement for generating custom exceptions. Furthermore, the article addresses recommended principles and best practices in exception management, aiming to support the development of robust, predictable applications that align with modern programming quality standards.
Keywords: exception handling mechanism, C# programming language, error handling, exceptions, types of exceptions.

Gestionarea excepțiilor în C# – o condiție esențială pentru dezvoltarea unor aplicații sigure

În orice proces de dezvoltare a aplicațiilor software, apariția erorilor în timpul execuției este inevitabilă. Indiferent de cât de bine este scris codul, există întotdeauna factori neprevăzuți care pot duce la situații de eroare: fișiere lipsă, date incorecte introduse de utilizator, conexiuni instabile la baze de date sau resurse indisponibile. Dacă astfel de situații nu sunt gestionate corespunzător, aplicația poate eșua. În asemenea condiții, pot apărea consecințe negative care influențează atât experiența utilizatorului, cât și stabilitatea aplicației.

Pentru a preveni aceste probleme, C# oferă un mecanism puternic și flexibil de tratare a excepțiilor, bazat pe instrucțiunile `try-catch-finally`. Acest model permite identificarea și gestionarea erorilor în mod controlat, prevenind oprirea neașteptată a

programului și permițând programatorilor să implementeze strategii eficiente de recuperare. Blocul `try` conține secvența de cod care poate genera o excepție în timpul execuției, catch captează și tratează acea excepție, iar `finally` este utilizat pentru a elibera sau închide resursele, garantând executarea acestuia indiferent dacă a avut loc sau nu o excepție [1].

Tratarea excepțiilor nu are ca scop doar prevenirea întreruperii execuției programului, ci contribuie, totodată, la creșterea securității aplicației, la facilitarea mentenanței și la asigurarea unei structuri de cod mai clare și mai ușor de înțeles. De exemplu, gestionarea corectă a erorilor într-o aplicație care interacționează cu o bază de date poate preveni pierderea datelor sau coruperea acestora. Totodată, un mecanism eficient de gestionare a excepțiilor permite înregistrarea detaliată a erorilor, ceea ce simplifică procesul de depanare și contribuie la o monitorizare mai precisă a comportamentului aplicației.

Gestionarea excepțiilor nu este doar un instrument practic, dar constituie și un element esențial al dezvoltării unei aplicații realizate într-un mod responsabil și profesionist.

În continuare, vom explora modul în care gestionarea erorilor poate contribui la crearea unor aplicații mai stabile, eficiente și ușor de întreținut.

Ce este o excepție în limbajul de programare C#?

În C#, o excepție este un eveniment de eroare care apare în timpul execuției unui program și care întrerupe fluxul normal de execuție al acestuia. Excepțiile sunt generate atunci când programul întâlnește o situație neașteptată sau eronată, cum ar fi:

- accesarea unui index inexistent într-o listă,
- împărțirea unui număr la zero,
- conversia incorectă a unui tip de date,
- deschiderea unui fișier care nu există,
- erori de rețea, de memorie etc.

Mecanismul excepțiilor în limbajul de programare C# implementează un sistem sigur de gestionare a excepțiilor, bazat pe clase derivate din clasa de bază *System.Exception* [1]. O excepție este un obiect care conține informații despre eroarea apărută, precum:

- mesajul de eroare (Message)
- sursa erorii (Source)
- stiva de apeluri (StackTrace)
- tipul excepției (Exception type)
- excepții interioare (InnerException) – în cazul erorilor în lanț

Excepțiile pot fi generate automat de sistem în momentul în care survine o eroare la rulare — de exemplu, în cazul unor excepții precum `DivideByZeroException` sau `NullReferenceException`. De asemenea, ele pot fi declanșate în mod explicit de

către programator prin utilizarea instrucțiunii `throw`, în scopul semnalării unor condiții excepționale specifice, definite în logica aplicației.

Exemplu:

```
    throw new InvalidOperationException("Operațiune invalidă.");
```

Tratarea excepțiilor

Pentru a trata o excepție și a evita oprirea neașteptată a aplicației, se folosește construcția:

```
try
{
    // Cod care poate genera excepții
}
catch (Exception ex)
{
    // Cod de tratare a erorii
}
finally
{
    // Cod care se execută întotdeauna (opțional)
}
```

Notă: - *Exception* este tipul obiectului care va fi capturat; - *ex* este numele variabilei care va stoca obiectul excepției capturate. Această variabilă poate fi utilizată ulterior în interiorul blocului catch pentru a accesa informații despre excepție (cum ar fi mesajul, sursa, stiva de apeluri etc.).

O excepție în C# este un mecanism elegant de gestionare a erorilor în timpul execuției. În loc ca aplicația să se oprească brusc, excepțiile permit tratarea controlată a problemelor, oferind programatorilor un mod clar și sigur de a răspunde la evenimente neprevăzute [2].

Această structură permite programului să gestioneze și să recupereze erorile, în loc să stopeze brusc programul.

- **try** – Blocul de cod în care poate apărea o eroare.
- **catch** – Prinde eroarea și o gestionează.
- **finally** –Se execută întotdeauna, indiferent dacă apare o eroare sau nu.

Blocul `try` reprezintă o structură de control utilizată în programare pentru a delimita o secvență de cod care poate genera erori în timpul execuției. Scopul său este de a permite identificarea și tratarea excepțiilor (adică a situațiilor neașteptate sau a erorilor de execuție) într-un mod controlat, fără ca programul să se oprească brusc.

Blocul `try` delimitează secvența de cod care prezintă potențialul de a genera o excepție în timpul execuției, cum ar fi accesarea unui fișier inexistent, conversii de date incorecte sau efectuarea unor operații matematice nevalide. De regulă blocul `try` este urmat de unul sau mai multe blocuri `catch`, care interceptează și gestionează excepțiile

apărute, și, opțional, de un bloc `finally`, care conține instrucțiuni ce sunt executate indiferent dacă s-a produs sau nu o excepție în timpul rulării codului.

Exemplu:

```
using System;
class Program
{
    static void Main()
    {
        try
        {
            int[] numere = { 1, 2, 3 };
            Console.WriteLine(numere[5]);
        }
        catch (IndexOutOfRangeException e)
        {
            Console.WriteLine("Eroare: Ai accesat un index care nu
există!");
        }
        finally
        {
            Console.WriteLine("Acest mesaj se va afișa mereu.");
        }
    }
}
```

În exemplul de program de mai sus, se încearcă accesarea celui de-al cincilea element din vectorul numit `numere`, deși acesta conține doar trei elemente, indexate de la 0 la 2. Această operațiune generează o excepție care este capturată de blocul `catch`, ce tratează eroarea și afișează un mesaj prietenos, prevenind astfel întreruperea bruscă a execuției. Blocul `finally` se execută în orice situație, indiferent dacă a fost sau nu declanșată o excepție [2].

Declanșarea explicită a unei excepții

În C#, instrucțiunea `throw` permite programatorului să declanșeze în mod explicit o excepție, în situațiile în care este necesară semnalarea unei situații anormale, o încălcare a unei reguli sau o eroare specifică în cadrul programului. Aceasta oferă un mecanism controlat pentru a opri execuția normală și a trata condițiile neprevăzute.

Exemplu:

```
using System;
class Program
{
    static void VerificaVarsta(int varsta)
    {
        if (varsta < 18)
        {
            throw new ArgumentException("Trebuie să ai cel puțin 18
ani.");
        }
```

```
        Console.WriteLine("Acces permis!");
    }

    static void Main()
    {
        try
        {
            VerificaVarsta(16);
        }
        catch (Exception e)
        {
            Console.WriteLine("Eroare:"+ e.Message);
        }
    }
}
```

Funcția `VerificaVarsta(int varsta)` are rolul de a verifica dacă valoarea furnizată pentru vârstă este mai mică de 18 ani. În cazul în care condiția este îndeplinită, instrucțiunea `throw` declanșează o excepție de tip `ArgumentException`, semnalând că valoarea este invalidă. Apelul funcției este realizat în cadrul blocului `try`, iar orice excepție generată este capturată în blocul `catch`, unde este tratată prin afișarea unui mesaj adecvat.

Blocurile `try-catch-finally`, instrucțiunea `throw` și mecanismele de gestionare a excepțiilor — sunt elemente fundamentale pentru dezvoltarea de aplicații sigure și ușor de întreținut.

Aceste concepte sunt utile pentru (1) prevenirea întreruperii neașteptate a programului; (2) facilitarea depănării sau a debuggingului; (3) îmbunătățirea securității aplicației; (4) creșterea clarității și organizării codului; (5) posibilitatea de a defini comportamente specifice pentru gestionarea situațiilor de eroare; (6) asigurarea eliberării resurselor. De asemenea, ele contribuie esențial la calitatea și stabilitatea aplicațiilor software, fiind considerate bune practici în programarea modernă. Fără un sistem eficient de tratare a excepțiilor, chiar și cele mai simple aplicații pot deveni vulnerabile, greu de întreținut și imprevizibile în comportament.

În continuare vom analiza mai detaliat ce înseamnă fiecare element din expresia `catch (Exception e)`.

- `Exception` este clasa de bază pentru toate excepțiile din C#.
- `e` este o variabilă de tip `Exception` care stochează informații despre eroare de exemplu: mesaj sau stack trace.
- Codul din `catch` se execută doar dacă apare o excepție în blocul `try`.

Exemplu:

```
using System;
class Program
{
    static void Main()
```

```
    {
        try
        {
            int x=10;
            int y=0;
            int rezultat=x/y;//Va genera o excepție (împărțire la
zero)
        }
        catch (Exception e)
        {
            Console.WriteLine("A apărut o eroare: " + e.Message);
        }
    }
}
```

Împărțirea la zero (x/y) generează o excepție (DivideByZeroException), iar `catch (Exception e)` prinde orice tip de excepție și afișează mesajul de eroare.

Utilizarea expresiei `catch (Exception e)` este adecvată în situațiile în care nu este cunoscut în prealabil tipul exact al erorii, dar se dorește interceptarea oricărei excepții apărute în timpul execuției, în special pentru a permite înregistrarea sau gestionarea centralizată a erorilor într-un mod generic, mai ales în contextul aplicațiilor de dimensiuni mari sau complexe [3].

Gestionarea controlată a erorilor prin identificarea excepțiilor specifice

Instrucțiunea `catch (Exception e)` ar trebui utilizată cu prudență, deoarece poate intercepta toate excepțiile, inclusiv pe cele neașteptate, ceea ce riscă să ascundă probleme critice care ar necesita o atenție specială. În plus, această abordare nu oferă informații detaliate despre natura exactă a erorii, ceea ce limitează posibilitățile de intervenție precisă; din acest motiv, este adesea recomandat să se folosească blocuri catch pentru tipuri specifice de excepții, care permit o tratare mai adecvată și mai transparentă a situațiilor de eroare.

Exemplu:

```
try
   {
   int[] numere = { 1, 2, 3 };
   Console.WriteLine(numere[5]);//Va genera IndexOutOfRangeException}
   catch (IndexOutOfRangeException e)
   {
     Console.WriteLine("Index invalid: " + e.Message);
   }
   catch (DivideByZeroException e)
   {
     Console.WriteLine("Împărțire la zero: " + e.Message);
   }
   catch (Exception e)
   {
     Console.WriteLine("Altă eroare: " + e.Message);
      }
```

Capturarea excepțiilor specifice permite un control mai precis asupra modului în care sunt tratate erorile în funcție de natura lor și contribuie la o structurare mai clară și previzibilă a codului sursă. Prin identificarea și tratarea distinctă a fiecărui tip de excepție, programatorul poate furniza răspunsuri adecvate fiecărei situații, ceea ce nu doar îmbunătățește stabilitatea aplicației, ci și facilitează întreținerea și depanarea acesteia. În plus, gestionarea diferențiată a excepțiilor reduce riscul de a ascunde erori critice și oferă o imagine mai exactă asupra comportamentului aplicației în condiții de execuție neprevăzute [4]. Instrucțiunea `catch (Exception e)` este utilă pentru gestionarea generală a erorilor, însă, în situațiile în care sunt cunoscute tipurile de excepții ce pot apărea, este recomandabilă utilizarea blocurilor catch specifice, deoarece acestea permit o tratare mai precisă și mai eficientă a problemelor întâlnite în timpul execuției.

După interceptarea unei excepții prin intermediul construcției `catch (Exception e)`, programatorul are posibilitatea de a analiza detalii despre eroare prin intermediul proprietății `e.Message`, utilizată adesea în scopuri de diagnosticare și afișare a mesajelor de eroare [5].

Relevanța proprietății `e.Message` în tratarea controlată a erorilor

În C#, `e.Message` este o proprietate a obiectului de tip `Exception` care returnează un text descriptiv ce oferă informații despre natura erorii apărute în timpul execuției programului. Această proprietate este utilă pentru diagnosticarea și înțelegerea cauzei unei excepții, deoarece oferă un rezumat concis al problemei, generat de sistemul .NET sau definit de programator în momentul în care excepția este aruncată.

Mesajul returnat de `e.Message` poate varia în funcție de tipul excepției și contextul în care aceasta a fost declanșată. De exemplu, o excepție de tip `FormatException` ar putea returna mesajul „Input string was not in a correct format”, indicând faptul că a eșuat o conversie de date.

Această proprietate este frecvent utilizată în blocurile `catch` pentru a afișa mesaje de eroare utile utilizatorului sau pentru a înregistra erorile într-un fișier de jurnal, oferind astfel un suport valoros pentru depanare și mentenanță.

Exemplu:

```
using System;

class Program
{
    static void Main()
    {
        try
        {
            int x=10;
            int y=0;
```

```
            int rezultat=x/y; // Generăm o excepție (împărțire la
zero)
        }
        catch (Exception e)
        {
           Console.WriteLine("A apărut o eroare:"+ e.Message);
        }
    }
}
```

Atunci când apare o eroare de tipul împărțire la zero, sistemul generează automat o excepție de tip `DivideByZeroException`. În acest caz, proprietatea `e.Message` returnează textul predefinit „Attempted to divide by zero.”, care reprezintă mesajul standard asociat acestei excepții.

Acest mesaj are rolul de a informa programatorul sau utilizatorul despre natura exactă a problemei apărute în timpul execuției. Deși mesajul este generic, el este suficient de clar pentru a indica sursa erorii, facilitând identificarea rapidă a cauzei și corectarea codului. În cadrul procesului de depanare sau jurnalizare a aplicației, accesarea `e.Message` devine astfel un instrument esențial pentru interpretarea corectă a excepției și pentru luarea unor măsuri corespunzătoare.

Exemplu cu alte tipuri de erori:

```
using System;

class Program
{
    static void Main()
    {
        try
        {
         int[] numere = { 1, 2, 3 };
         Console.WriteLine(numere[5]);/*Generăm
                    IndexOutOfRangeException*/
        }
        catch (Exception e)
        {
            Console.WriteLine("Eroare: " + e.Message);
        }
    }
}
```

Rezultatul afișat în momentul executării codului este: „Eroare: Index was outside the bounds of the array.”, ceea ce indică faptul că programul a încercat să acceseze un element aflat în afara limitelor definite ale unui tablou (array).

Acest mesaj este furnizat prin intermediul proprietății `e.Message`, care, în cazul excepției `IndexOutOfRangeException`, returnează un text standardizat generat de sistemul .NET pentru a semnala că indicele utilizat depășește dimensiunea reală a colecției.

Această informație este deosebit de utilă în procesul de depanare, deoarece permite programatorului să identifice rapid eroarea logică din cod – de exemplu, o iterație incorectă sau o calculare greșită a unui index. Utilizarea `e.Message` în astfel de contexte asigură o comunicare clară a naturii excepției, contribuind la o gestionare eficientă a erorilor și la menținerea stabilității aplicației.

Pe lângă proprietatea `e.Message`, care oferă o descriere generală a erorii apărute, obiectul de tip `Exception` în C# pune la dispoziția programatorului și alte proprietăți esențiale pentru analiza detaliată și tratarea eficientă a excepțiilor [6]. Acestea contribuie semnificativ la procesul de depanare și la dezvoltarea unor aplicații sigure și ușor de întreținut.

- `e.StackTrace`

Această proprietate returnează un șir de caractere care conține traseul stivei de apeluri în momentul în care excepția a fost generată. Este deosebit de valoroasă pentru identificarea exactă a locului din cod unde a apărut eroarea, incluzând metodele apelate și fișierele sursă, dacă sunt disponibile. Proprietatea `StackTrace` este frecvent utilizată pentru înregistrarea detaliată a erorilor, oferind dezvoltatorilor informații precise despre contextul în care a avut loc excepția, facilitând astfel identificarea rapidă și corectarea problemei.

- `e.InnerException`

În situațiile în care o excepție este rezultatul unei alte excepții anterioare, `InnerException` oferă acces la excepția inițială care a declanșat eroarea curentă. Această proprietate este deosebit de utilă în situații mai complicate, în care o eroare inițială duce la apariția altor erori, formând un lanț de probleme între metode sau componente. Prin accesarea `InnerException`, programatorul poate descoperi care a fost cauza principală a excepției, chiar dacă aceasta a fost „ascunsă” sub alte mesaje de eroare generate ulterior. Astfel, se obține o imagine mai clară asupra originii reale a problemei.

- `e.GetType()`

Această metodă furnizează informații despre tipul specific al excepției care a fost generată, cum ar fi, de exemplu, `System.NullReferenceException` sau `System.IndexOutOfRangeException`. Identificarea precisă a tipului de excepție joacă un rol esențial în procesul de gestionare a erorilor, deoarece permite diferențierea între diverse categorii de excepții și aplicarea unor strategii de tratare adecvate, adaptate nivelului de gravitate sau contextului în care acestea apar.

Exemplu cu mai multe detalii despre eroare:

```
try
{
    int num = int.Parse("abc"); // Generăm FormatException
}
catch (Exception e)
```

```
{
    Console.WriteLine("Tip eroare: " + e.GetType());
    Console.WriteLine("Mesaj eroare: " + e.Message);
    Console.WriteLine("Detalii eroare: " + e.StackTrace);
}
```

Executarea acestui cod va conduce la afișarea unor informații detaliate despre excepția apărută în timpul rulării aplicației. În primul rând, va fi afișat tipul exact al erorii, în acest caz `System.FormatException`, care indică faptul că s-a încercat conversia unui șir de caractere într-un format numeric invalid.

Apoi, va fi prezentat mesajul de eroare furnizat de sistem, respectiv: „Input string was not in a correct format.”, mesaj care explică în termeni simpli cauza problemei – o eroare de formatare a datelor.

În cele din urmă, codul va afișa detalii suplimentare despre traseul execuției (`StackTrace`), oferind o imagine clară asupra locației exacte în cod unde a fost generată excepția. Aceste informații sunt esențiale pentru depanare, întrucât permit dezvoltatorului să identifice rapid sursa problemei și să intervină eficient pentru remedierea acesteia.

Prin urmare putem concluziona că proprietatea `e.Message` reprezintă unul dintre cele mai accesibile și utile instrumente oferite de clasa `Exception` în C#. Ea returnează un mesaj descriptiv care explică natura erorii apărute în timpul execuției programului, fiind esențială atât în etapa de dezvoltare, cât și în cea de testare [6].

Utilizarea `e.Message` este recomandată în special pentru afișarea mesajelor de eroare într-o formă prietenoasă pentru utilizator, permițând în același timp programatorului să înțeleagă rapid cauza problemei. Această proprietate este extrem de valoroasă în procesele de debugging, contribuind la o remediere mai eficientă a erorilor.

Pentru situații în care este necesară o analiză mai detaliată a excepției, `e.Message` poate fi completată cu alte instrumente precum `e.StackTrace,` care oferă informații despre traseul stivei de apeluri, sau `e.GetType()`, care permite identificarea tipului specific de excepție. Împreună, aceste proprietăți formează un set esențial pentru diagnosticarea corectă și tratarea profesionistă a erorilor în aplicațiile dezvoltate în C#.

Concluzii

Gestionarea corectă a excepțiilor reprezintă un element fundamental în procesul de dezvoltare software, influențând în mod direct stabilitatea, securitatea și ușurința în mentenanță a aplicațiilor. Prin implementarea adecvată a mecanismelor `try-catch-finally`, dezvoltatorii pot evita întreruperile neașteptate ale execuției, pot facilita tratarea controlată a situațiilor de eroare și pot contribui la menținerea unei experiențe coerente și sigure pentru utilizatori. În plus, o strategie bine concepută de gestionare a

excepțiilor permite nu doar prevenirea defectelor critice, ci și identificarea rapidă a acestora, susținând astfel dezvoltarea de aplicații stabile și performante.

O abordare corectă a gestionării excepțiilor depășește simpla interceptare a erorilor și implică dezvoltarea unor soluții eficiente de remediere, menite să reducă impactul negativ asupra funcționării sistemului. În acest context, este important să se evite utilizarea excesivă a capturilor generice, precum `catch(Exception e)`, și să se acorde prioritate identificării și tratării excepțiilor specifice, pentru a permite o diagnosticare mai precisă și o intervenție mai bine direcționată.

Astfel, gestionarea erorilor nu trebuie percepută doar ca o soluție de evitare a blocajelor critice, ci ca o componentă strategică în optimizarea performanței aplicațiilor. O excepție tratată corespunzător poate deveni un punct de sprijin pentru îmbunătățirea stabilității și consistenței funcționale a aplicației și pentru crearea unei experiențe mai coerente și mai plăcute pentru utilizator.

Articol realizat în cadrul proiectului de cercetări științifice „Metodologia implementării TIC în procesul de studiere a științelor reale din perspectiva conceptului STEAM și Inteligenței Artificiale”, codul 040101, din cadrul Programului instituțional de cercetare (2024-2027), aprobat prin Ordin MEC nr. 102 din 01.02.2024

Bibliografie

1. ALBAHARI, J., ALBAHARI, B. *C# 10 in a Nutshell: The Definitive Reference*. O'Reilly Media, 2022.
2. TROELSEN, A., JAPIKSE, P. *Pro C# 9 with .NET 5: Foundational Principles and Practices in Programming*. Apress, 2021.
3. SKEET, J. *C# in Depth* (4th Edition). Manning Publications, 2019.
4. RICHTER, J. *CLR via C#* (4th Edition). Microsoft Press, 2017.
5. Microsoft Docs. Exception handling (C# Programming Guide). 2023. https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/exceptions/
6. LIPPERT, E. *Vexing exceptions*. 2008. https://ericlippert.com/2008/09/10/vexing-exceptions/

Recepționat / Received: 29.04.2025
Acceptat / Accepted: 19.06.2025
Email: gasnas.ala@upsc.md